SUGAR: A Graph Database Fuzzy Querying System

Olivier Pivert, Olfa Slama, Grégory Smits, Virginie Thion

SUGAR: A Graph Database Fuzzy Querying System

Olivier Pivert, Olfa Slama, Grégory Smits, Virginie Thion
Rennes 1 University / IRISA – Lannion, France
Email: {Olivier.Pivert, Olfa.Slama, Gregory.Smits, Virginie.Thion}@irisa.fr

***Abstract*—Graph databases have aroused a large interest in the last years thanks to their large scope of potential applications. Defining a language allowing a flexible querying of graph databases may greatly improve usability of data. In this paper, we present a system for querying graph databases in a flexible way. The preferences are based on fuzzy set theory and may concern i) the content of the vertices and ii) the structure of the graph.**

I. Introduction

Much work has been done about fuzzy querying of *relational* databases (see e.g. [1]). However, even though relational databases are still widely used, the need to handle *complex* data has led to the emergence of other types of data models. In the last few years, a new concept has started to attract a lot of attention in the database world, namely that of *graph databases* (see e.g. [2]), whose basic purpose is to efficiently manage networks of entities where each node is described by a set of characteristics (e.g. a set of attributes), and each edge represents a link between entities. Such a database model has many potential applications, e.g. for modeling social networks, RDF data, cartographic databases, bibliographic databases, etc. This model may be extended into the notion of a *fuzzy graph database* where a degree may be attached to edges. This degree may express the "intensity" of any kind of gradual relation between two nodes (e.g. *likes*, *is friends with*, *is about*). Graph databases, which may be fuzzy or not, raise new challenges in terms of flexible querying since two aspects may be involved in the preferences that a user may express: i) the content of the nodes and ii) the graph structure.

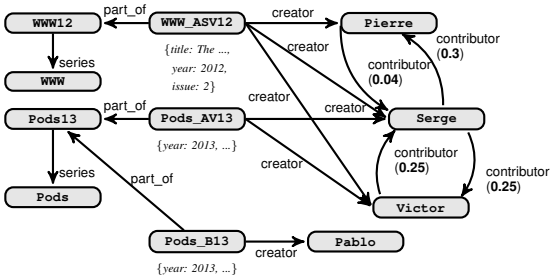

Fig. 1. A fuzzy data graph $\mathcal{DB}$ derived from an excerpt of DBLP data

We present SUGAR, a system that makes it possible to query a fuzzy graph database in a flexible way. This system is built on the Neo4j Graph Database Management System [3] and implements the FUDGE language introduced in [4].

II. Graph Database Fuzzy Querying

The last decade has witnessed a growing interest in preference queries (see e.g. [1]). Motivations for introducing preferences inside database queries are manifold. First, it has appeared to be desirable to offer more expressive query languages that can be more faithful to what a user intends to say. Second, the introduction of preferences in queries provides a basis for rank-ordering the retrieved items, which is especially valuable in case of a large set of items satisfying a query. Third, a classical query may also have an empty set of answers, while a relaxed version of the query may be matched by some items. In the graph database context, the need for having flexible querying tools is even more acute as users are in general unfamiliar with the representation of the data [2]. In the following, we focus on the fuzzy-set based approach to preference queries, which is founded on the use of fuzzy set membership functions that describe the preference profiles of the user on each attribute domain involved in the query.

Fuzzy set theory allows to model classes or sets whose boundaries are not clear-cut. For such sets, the transition between full membership and full mismatch is gradual rather than crisp (non fuzzy). Typical examples of such fuzzy classes are those described using adjectives of the natural language, such as *young, short, cheap*, etc. Formally, a fuzzy set F on a referential U is characterized by a membership function $\mu_F : U \rightarrow [0, 1]$ where $\mu_F(u)$ denotes the grade of membership of u in F. In particular, $\mu_F(u) = 1$ reflects full membership of u in F, while $\mu_F(u) = 0$ expresses absolute non-membership. When $0 < \mu_F(u) < 1$, one speaks of partial membership. Two crisp sets are of particular interest when defining a fuzzy set F: the core $C(F) = \{u \in U \mid \mu_F(u) = 1\}$, which gathers the *prototypes* of F, and the support $S(F) = \{u \in U | \mu_F(u) > 0\}$. In practice, the membership function associated with F is often of a trapezoidal shape. Then, F is expressed by the quadruple (A, B, a, b) where is $C(F) = [A, B]$ and $S(F) = [A-a, B+b]$.

III. The FUDGE language

The FUDGE language [4] is based on the CYPHER query language, used for querying graph databases in a crisp way in the Neo4j graph database management system [3]. FUDGE relies on a formal algebra defined in [4].

We focus here on selection queries in which fuzzy preferences may appear. A FUDGE selection query is composed of: (1) a list of `DEFINE` clauses for fuzzy term declarations. If a fuzzy term `fterm` is defined by a trapezoidal function with the four positions (abscissa) `A-a`, `A`, `B` and `B+b`, then the clause has the form `DEFINE fterm AS (A-a,A,B,B+b)`. If `fterm` is a decreasing membership function, then the clause has the form `DEFINEDESC fterm AS (B,B+b)` (and there is the corresponding `DEFINEASC` clause for increasing functions). (2) a `MATCH` clause of the form `MATCH pattern WHERE condition` where `pattern` is a fuzzy graph pattern and `condition` is the possibly compound condition attached to the pattern. A graph pattern is defined *à la ASCII art*, in the CYPHER way, where the graphic symbol `( )` denotes a node, which may contain information

of the form `query_variable:Type` concerning this node. The symbol `-[form]->` denotes a connection i.e. either an edge where `form` may denote the label of the edge, or a path where `form` is a fuzzy regular expression that the path has to satisfy.

The FUDGE Query 1 aims to retrieve information concerning authors (`au2`) who have, among their *close* contributors, an author (`au1`) who published a paper (`ar1`) in `WWW` and also published a paper (`ar2`) in `Pods` *recently* (`ar2.year IS recent`). The `DEFINE` clauses define the fuzzy terms *short* and *recent*.

```
DEFINEDESC short AS (3,5), DEFINEASC recent AS (2010,2014)
MATCH
  (ar1:Article)-[part_of]->()->[series]->(s1),
  (ar2:Article)-[part_of]->()->[series]->(s2),
  (ar1)-[:creator]->(au1:Author),
  (ar2)-[:creator]->(au1:Author),
  (au1)-[(contributor+)|Length IS short]->(au2:Author)
WHERE s1.id=WWW AND s2.id=Pods AND ar2.year IS recent
```

Query 1. A FUDGE query

As illustrated here, fuzzy conditions may concern either the content of vertices and edges, e.g. the year of an article may be *recent*, or a path, e.g. a path going from an author to another one is required to be *short*[1].

IV. The SUGAR software

The SUGAR software implements the FUDGE language. It is built from the Neo4j REPL Console Rabbithole [6] supporting CYPHER queries. SUGAR is a software add-on composed of two modules, which interact with a Neo4j crisp engine (see Fig. 2): the *Transcriptor* module and the *Score Calculator* one. The former aims to translate a FUDGE query into a (crisp) CYPHER one that retrieves all needed data. This query is then sent to the crisp Neo4j engine. Then the *Score Calculator* module extracts answers and calculates the satisfaction degree associated with each answer returned by the crisp engine, and ranks the answers.

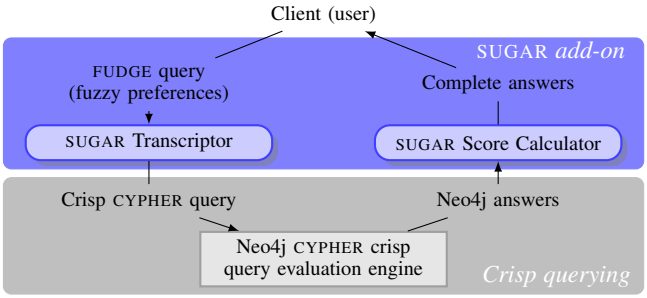

Fig. 2. SUGAR software architecture

SUGAR proposes a user-friendly GUI, which is an extension of the Rabbithole one. It is composed of (i) a frame for visualizing the graph database and the results of a query and (ii) an input field frame for entering and executing a FUDGE query. SUGAR also provides logs that trace the evaluation in an associated console, providing each intermediate result of the execution process: the crisp CYPHER query obtained after the transcription stage (output of the *Transcriptor module*), the result of the crisp query evaluation (an intermediate result containing additional information needed for the score calculation), and the final result obtained after the score calculation (output of the *Score Calculator* module). The logs also provide the execution time associated with each stage of the evaluation.

[1]The definition of the `length` of a path is a fuzzy graph notion defined by [5]: $Length(p) = \sum_{i=1}^{n} \frac{1}{\rho(x_{i-1}, x_i)}$ where $\rho(x, y)$ is the degree attached to the edge (x, y).

V. Demonstration scenario

During the demonstration, the user is invited to interact directly with SUGAR by entering FUDGE queries, either his/her own queries, or queries of a predefined set. This set mixes essential kinds of graph queries ([7], [8]), i.e. attribute searching queries, node/edge adjacency queries, fixed-length paths queries, reachability queries and graph pattern matching queries, including fuzzy preferences or not. These queries propose, for some simple and relevant intent use cases, a crisp version and a version that includes fuzzy preferences. Above all else, these queries **show the interest of introducing flexibility**, facilitating i) **query formulation** by using intuitive fuzzy terms and ii) **result readability** as the satisfaction degree associated with each answer allows to rank the answers.

Then, the user may examine the answers and the associated logs, not only to see the result of a query, but also to understand the evaluation process implemented in the software for the evaluation of a FUDGE query, and to make some interesting observations. One is the **conciseness** of the FUDGE language. Indeed, the crisp CYPHER query obtained after the transcription stage is really complex wrt. the FUDGE one. Concerning the **cost** of introducing flexibility, the user will observe that the transcription step is only a small part of the overall evaluation process as it only consists of a linear parsing, and transformation when needed, of the FUDGE query. A similar observation concerns the score calculation step, which only consists of a linear parsing of the set of the crisp query answers for calculating the satisfaction degree associated with each of them, and a classical ranking of the answers according to the degrees. In the running example, the cost of introducing flexibility is less than 3% of the total cost.

Because of space limitation, we do not insert software screenshots here but the SUGAR system can be downloaded at www-shaman.irisa.fr/fudge-prototype, which also proposes complementary screenshots.

Acknowledgement: This work has been partially funded by the French DGE (Direction Générale des Entreprises) under the project ODIN (Open Data INtelligence).

References

[1] O. Pivert and P. Bosc, *Fuzzy Preference Queries to Relational Databases*. Imperial College Press, 2012.

[2] P. T. Wood, "Query languages for graph databases," *SIGMOD Record*, vol. 41, no. 1, pp. 50–60, 2012.

[3] "Neo4j web site," www.neo4j.org, accessed in 2016.

[4] O. Pivert, V. Thion, H. Jaudoin, and G. Smits, "On a fuzzy algebra for querying graph databases," in *Proc. of IEEE ICTAI*, 2014, pp. 748–755.

[5] A. Rosenfeld, "Fuzzy graphs," in *Fuzzy Sets and their Applications to Cognitive and Decision Processes*. Academic Press, 1975, pp. 77–97.

[6] "RabbitHole Console," http://neo4j.com/blog/rabbithole-the-neo4j-repl-console, accessed in 2016.

[7] R. Angles, "A comparison of current graph database models," in *Proc. of ICDE Workshops*, 2012, pp. 171–177.

[8] R. Angles, P. Barceló, and G. Rios, "A Practical Query Language for Graph DBs," in *Proc. of Alberto Mendelzon Intl. Workshop on Foundations of Data Management (AMW)*, 2013.